IP Support

Internet protocols for local stations

July 12, 1996

Introduction

In recent years, the internet protocol standard has grown by leaps and bounds. Because support for it is included with every workstation, it is natural to consider providing this support for the local station systems, in order to make it easier to write support for data requests and settings to a control system. This note is a working document that assumes some TCP/IP familiarity.

Note that internet protocol support is only a beginning for a host; the data request protocols must be built on top of any internet protocol used. The fundamental node-node communications is supported by IP, Internet Protocol. In order to make the network accessible to a user, an entity *within* a node, a higher level is needed. Above IP, the TCP/IP protocol suite includes two basic types of data communications: stream-oriented (TCP) and datagram-oriented (UDP). Each of these types is suited for different applications, yet higher level protocols.

The stream-oriented protocol (TCP) refers to a byte stream of data. Network messages that support the byte stream have no built-in boundaries at all, so a user of TCP must build that into the higher layers. It does, however, insure that the byte stream arrives in the same order as the byte stream that was sent, by use of sliding window acknowledgements. If it were important to support Telnet, FTP, or SMTP with the local control stations, then TCP support would be required.

The datagram-oriented protocol (UDP) is used to transport a record, called a datagram, across a network. It is sent on a best efforts basis, with no guarantee that the datagram arrives at the receiving node nor that a succession of data grams arrives in order. For a control system, which is fundamentally record-oriented, the datagram approach has appeal. The lack of a guarantee that the datagram reached its destination node is mitigated by the reply to a data request, or the acknow ledg ment to a setting. The lack of guarantee of the order is really only a problem for very large internets and is unlikely to be a problem in practice within a single laboratory. (If one were attempting to do 15 Hz control from Sweden, for example, there might be need for more concern.) Protocols that expect to use UDP are NFS, RPC, SNMP, and TFTP.

In view of the above arguments, assume that a data request/setting protocol is built on top of UDP, the datagram protocol.

Token ring frame format for IP

What is the frame format of IP datagrams on the token ring network? According to RFC-1042, the format uses the SNAP variation of the 802.2 header. This means that the DSAP and SSAP are $AA, the control byte is $03 (UI), the next three bytes are the organization code $000000, and the last two bytes are the same as the Ethernet type word. For IP frames, this is $0800. For ARP, it is $0806.

IP datagrams

What does the IP layer do? Its job is to send a datagram, limited to 64K bytes in length, to another node across an internet. This might imply that fragmen tation would occur. So the IP Task must support fragmentation and reassembly. In the current local station software, there is convenient support for frame *reception* passed to a receiving task. If no fragmentation takes place, then the current support for network messages largely takes care of itself, assuming that the protocols within the UDP header are the same as those handled now. If fragmentation occurs, so that a fragment is received that does not represent the entire datagram, then the fragment must be copied into a datagram buffer of larger size. This will make it slower, of course, but is probably unavoidable. When the complete datagram has been assembled, it can be passed to the destination task. As a first step in support, one could ignore fragmen tation on the token ring network. Data requests/replies have, until now, always been limited to about 4K bytes, the maximum frame size for token ring. (Since this was first written, fragmentation support was added 6/23/92.)

For frame *transmission*, there must be some means of denoting that IP packaging is required. This must include a way to keep the port# of the requester as well as the requesting node. A pseudo-node# can be used as the source node of the request that can help recover the original requester's UDP source port#.

The idea here is that support for the present suite of protocols remains and is only enhanced by the addition of IP and UDP support. These provides an optional wrapper for the usual protocols. To support both Classic and Acnet-header-based protocols, two different well-known UDP server ports are used.

Address resolution

What about ARP? The Address Resolution Protocol is used to find the hardware address that corresponds to a given IP address in the case that the IP address is on the same subnetwork. This request is broadcast with a special value for the type word ($0806), in hopes that a node will answer with the hardware address that corresponds to the given IP address.

What about ARP processing? If we need to send an ARP, it's one more comp lication, because it means that the frame cannot be transmitted until a reply is often worry about sending a message to a hitherto unknown node.

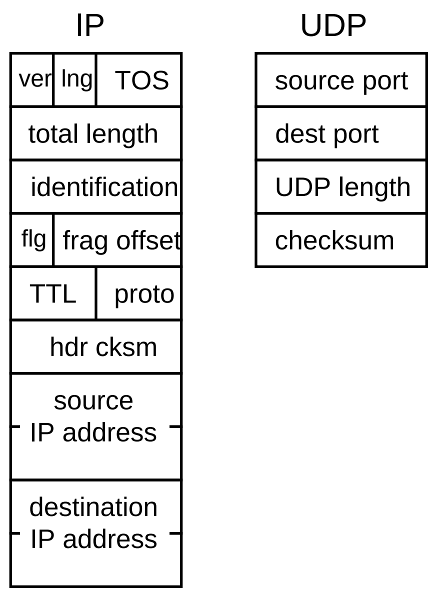
The ARP table entries are usually timed out, in order to accommodate changes in network addresses. If this is done, it will become important to support queuing of datagrams awaiting ARP replies. At first, we can keep ARP table entries forever and not support datagram queuing.

IP addresses

When a message is received, a node# must somehow be assigned to it. If the requesting node does not use the `40020000ttnn` convention, then it might be an Ethernet console with node# `08xx` using the `55002000yy17` convention, in which the yy is the bit-reversed value of `nn`. There is a table of Ethernet addresses that correspond to `0800–08EF`. If we can use some of that space for dynamically assigned IP requesters, it would provide a natural place to store the hardware address. But what about the IP address? When NetXmit has a `08zz` node#, it can find the right hardware address, say, but where can it find the IP address? Of course, it was in the request message, but where can it be stored for later retrieval? A different table should be used for this, one that is kept in non-volatile memory and filled whenever a frame is received from an IP address.

When a frame is received that uses IP or ARP protocols, capture the hardware address from the frame header and the IP address from the IP header and the UDP port# from the UDP header. Search the table for a match on these values. If none is found, install a new entry. Assign an *internal* node# and replace the source node# in the acnet header with it. In this way, a request message can be processed normally and the reply queued to the network using this internal node# as a destination. The NetXmit logic can use it as a signal to send an IP datagram and to recover the other information for building the frame header. See the document IPARP Table for more discussion on the table's implementation.

The IP and UDP headers have the following formats:

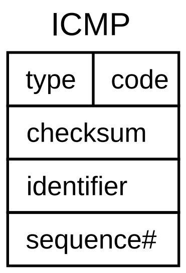


The *ver* is the IP protocol header version number $04, the *lng* is the number of longwords in the header, between 5 and 15, but usually 5. The *type-of-service* byte can be ignored initially and built as a constant (0) for transmission. The *total length* is the #bytes in the datagram, including both the IP header and the rest of the datagram. The *identification* is a word that is a sequence# of IP datagram sent by the source node. It has the same value in all fragments of a datagram. The *flg* bits and *frag offset* are used for fragmentation. The *don't fragment* flag bit prevents a gateway from forwarding a fragmented datagram. A *more fragments* flag bit indicates that this fragment is not the last one. It is only when a fragment of a datagram that contains the *more fragments* bit clear that IP learns the length of the entire datagram.

ICMP support

This is the error reporting mechanism. It is based upon IP just as UDP is, but it is a required part of IP support. It provides a "ping" echo service, for one. Error messages should be directed to the original source node in general, since the route taken by the message in error is not available. Care should be taken to use ICMP error replies only in situations specified by RFC-1122 recommendations.

The format of the ICMP message is as follows:



Network messages in local station

Local station network handling is a higher level of support than either IP or UDP, in the sense that it is message-based and not frame-based. An application using the network routines does not see the frame boundaries, although it can flush the network queue to force a frame boundary. It usually deals with messages that it receives from (or queues to) the network. All current network-related applications have this message-oriented view. Also, they can generate either Classic protocol messages or Acnet protocol messages. It is desirable to *not* break this mechanism by the intro duction of IP support. This means that frame transmission using IP datagram frames must be automatically determined by NetXmit logic, which extracts all queued network messages, combines them into frames and hands them over to the token ring chipset hardware.

Frame formats used by local station

How can NetXmit decide what type of frame header to use? The difference between Classic and Acnet is decided by the memory block type# that holds the queued message. But we also need to communicate using ARP and ICMP, so these may require an additional memory block type# that can be queued to the network that would cause NetXmit to formulate those frames appropriately. Also, one must insure that ARP and ICMP messages are not combined with others in a common frame, as no host will expect it. For the UDP frames we are discussing, multiple messages *can* be supported in the same frame because, to a host that receives such a frame, it is only a single message sent to a server UDP port. The port handling logic will identify the messages of, say, Acnet format and dispatch them to the appropriate network tasks.